TECNOLÓGICO DE ESTUDIOS SUPERIORES DE CHALCO
INGENIERÍA EN SISTEMAS COMPUTACIONALES

Sistema de Gestión Hotelera "StayEasy"

ANZURES CAMPOS JUAN ESTEBAN
ARELLANO PALACIOS DANIEL
FLORES MARTINEZ IAN ALEXIS
JURADO DELGADILLO MIGUEL ANGEL
REYES ARGUELLO JUAN FERNANDO
VILLEGAS RAYMUNDO BRAYAN URIEL

M. EN C.C. NIZA LUCERO ALPIZAR MARTINEZ

22 DE SEPTIEMBRE DE 2025

Introducción / Resumen Ejecutivo

El proyecto "StayEasy" consiste en el diseño y desarrollo de un Sistema de Gestión Hotelera (PMS, por sus siglas en inglés *Property Management System*) robusto y escalable. El propósito principal es centralizar y automatizar las operaciones clave de un hotel, tales como la gestión de reservaciones, el registro de huéspedes (check-in/check-out), la administración de habitaciones y la facturación.

El contexto del problema a resolver radica en las ineficiencias y errores comunes derivados de la gestión manual o el uso de sistemas obsoletos y fragmentados. Muchos hoteles pequeños y medianos todavía dependen de hojas de cálculo, registros en papel o software limitado, lo que provoca problemas como el overbooking, la pérdida de información del cliente, procesos de check-in lentos y una deficiente asignación de tareas de limpieza y mantenimiento. "StayEasy" busca ofrecer una solución tecnológica integrada que optimice los recursos, mejore la experiencia del huésped y facilite la toma de decisiones estratégicas para la administración del hotel.

Justificación y Alcance

Justificación

La implementación de un sistema de gestión hotelera es fundamental en la industria de la hospitalidad moderna. Un PMS eficiente no solo agiliza las operaciones diarias, sino que también impacta directamente en la rentabilidad y la satisfacción del cliente. Según un análisis de la industria, los sistemas de gestión hotelera mejoran la eficiencia operativa hasta en un 25% y aumentan las reservaciones directas al ofrecer una visión en tiempo real de la disponibilidad (Oracle, 2023). Este proyecto es importante porque aborda una necesidad crítica del mercado, desarrollando una herramienta que permite a los hoteles competir eficazmente, reducir la carga de trabajo administrativo y enfocarse en proporcionar un servicio de mayor calidad a sus huéspedes. La centralización de datos también genera reportes valiosos que apoyan la planeación estratégica y financiera.

Alcance Preliminar

Funcionalidades que se desarrollarán (In-Scope):

- **Módulo de Reservaciones:** Crear, consultar, modificar y cancelar reservaciones. Incluirá un calendario visual de ocupación.
- **Módulo de Recepción (Front Desk):** Gestionar el proceso de check-in y check-out de los huéspedes.
- **Módulo de Gestión de Habitaciones:** Administrar el estado de las habitaciones (disponible, ocupada, en limpieza, en mantenimiento).
- **Módulo de Gestión de Huéspedes:** Mantener una base de datos de los clientes con su historial de estancias.
- **Módulo de Facturación Básica:** Generar facturas simples por estancia, con cargos de alojamiento.
- **Autenticación y Roles de Usuario:** Sistema de inicio de sesión para el personal del
  hotel con roles definidos (ej. Recepcionista, Administrador).

Funcionalidades que NO se desarrollarán (Out-of-Scope):

- Integración con canales de venta de terceros (OTAs como Booking.com, Expedia).
- Módulo de punto de venta (POS) para restaurantes o tiendas del hotel.
- Sistema de gestión de inventario.
- Funcionalidades avanzadas de marketing y gestión de la relación con el cliente (CRM).

- Aplicación móvil para el personal de limpieza (housekeeping).
- Módulo de contabilidad avanzada.

Objetivos

- **Objetivo General:**
  - Desarrollar un prototipo funcional de un sistema de gestión hotelera ("StayEasy") que automatice los procesos de reservación, recepción y administración de habitaciones para un hotel ficticio de tamaño mediano.

- **Objetivos Específicos:**

1. Diseñar e implementar una base de datos relacional para almacenar de forma segura y eficiente la información de huéspedes, habitaciones, reservaciones y facturas.

2. Construir un módulo de reservaciones que permita al personal del hotel gestionar el ciclo de vida completo de una reserva en menos de 3 minutos por operación.

3. Desarrollar una interfaz de usuario web intuitiva y responsiva que facilite al personal de recepción realizar los procesos de check-in y check-out de manera ágil, reduciendo el tiempo promedio por huésped en un 40% en comparación con un sistema manual.

Análisis de Requerimientos Iniciales

Todos los requisitos que están en la sección, tanto las cosas que el programa tiene que hacer (los RF, como registrar una reserva, dar la habitación o cobrar la cuenta) como qué tan bien las debe hacer (los RNF, como que sea súper rápido, seguro y fácil de usar), se obtuvieron de una manera muy práctica: preguntando y simulando. Primero, definimos el alcance, es decir, pusimos en una lista de qué tareas del hotel sí va a cubrir el sistema para resolver los dolores de cabeza de la gestión manual (como el overbooking). Esto nos dio la lista de las funciones clave. Luego, definimos el nivel de calidad basándonos en nuestras metas: por ejemplo, si queremos que el check-in sea 40% más rápido, eso nos obliga a exigir que la interfaz sea muy intuitiva y que el rendimiento sea veloz. Además, para que el sistema no se convierta en un dolor de cabeza en el futuro, decidimos cómo construirlo internamente de una forma que sea fácil de modificar o arreglar sin tener que desarmar todo. Y por último, pensamos en los posibles desastres (como una caída del sistema o la pérdida de datos) y así nacieron los requisitos obligatorios de seguridad y las copias de respaldo, garantizando que el programa sea confiable y esté preparado para lo suceda.

Requisitos Funcionales		
ID	Descripción	Impacto
RF-001	El sistema debe permitir al usuario recepcionista crear una nueva reservación especificando fechas de entrada/salida, tipo de habitación y datos del huésped	CRÍTICO
RF-002	El sistema debe mostrar un calendario o dashboard visual con la ocupación de las habitaciones por fecha.	CRÍTICO
RF-003	El sistema debe permitir modificar los detalles de una reservación existente (ej. cambio de fechas).	IMPORTANTE

RF-004	El sistema debe permitir cancelar una reservación, liberando la habitación para esa fecha.	IMPORTANTE
RF-005	El sistema debe permitir registrar la llegada de un huésped (check-in), cambiando el estado de la habitación a "Ocupada".	CRÍTICO
RF-006	El sistema debe permitir registrar la salida de un huésped (check-out), generando una factura y cambiando el estado de la habitación a "En Limpieza".	IMPORTANTE
RF-007	El sistema debe permitir registrar y consultar la información de los huéspedes.	IMPORTANTE
RF-008	El sistema debe permitir al usuario administrador cambiar el estado de una habitación manualmente (ej. a "Mantenimiento").	MEDIA
RF-009	El sistema debe requerir autenticación (usuario y contraseña) para acceder a sus funcionalidades.	IMPORTANTE
RF-010	El sistema debe permitir al usuario administrador	CRÍTICO

	generar un reporte básico de ocupación por rango de fechas.	

Requisitos No Funcionales		
ID	Descripción	Impacto
RNF-001 (Usabilidad)	La interfaz de usuario debe ser intuitiva, permitiendo que un nuevo recepcionista aprenda a realizar las operaciones básicas en menos de una hora de capacitación.	IMPORTANTE
RNF-002 (Rendimiento):	Las consultas de disponibilidad de habitaciones para un rango de fechas deben ejecutarse y mostrar resultados en menos de 2 segundos.	MEDIA
RNF-003 (Seguridad)	La información personal de los huéspedes debe estar encriptada en la base de datos. El acceso a las funcionalidades debe estar restringido por roles.	IMPORTANTE
RNF-004 (Disponibilidad)	El sistema deberá tener una disponibilidad del 99.5% durante el horario operativo del hotel.	IMPORTANTE

RNF-005 (Compatibilidad)	La aplicación web debe ser compatible con las últimas dos versiones de los navegadores Google Chrome, Mozilla Firefox y Microsoft Edge.	MEDIA
RNF-006 (Escalabilidad)	El sistema debe ser capaz de soportar un aumento del 50% en el número de usuarios concurrentes y el volumen de datos de reservas sin degradación significativa del rendimiento.	IMPORTANTE
RNF-007 (Mantenibilidad)	El código fuente del sistema debe seguir estándares de codificación y estar documentado para facilitar futuras modificaciones y mejoras.	MEDIA
RNF-008 (Portabilidad)	El sistema debe poder ser desplegado en diferentes entornos de servidor con cambios mínimos en la configuración.	MEDIA
RNF-009 (Integridad de Datos)	El sistema debe asegurar la consistencia y exactitud de los datos en todas las operaciones, especialmente en la gestión de reservas y facturación.	CRÍTICO
RNF-010 (Respaldo y Recuperación)	El sistema debe contar con un mecanismo automático de respaldo de datos diario y un procedimiento documentado para la recuperación ante	IMPORTANTE

	desastres.	

Arquitectura de Software: Modelo Hexagonal (Ports and Adapters)

Principios Fundamentales y Motivación

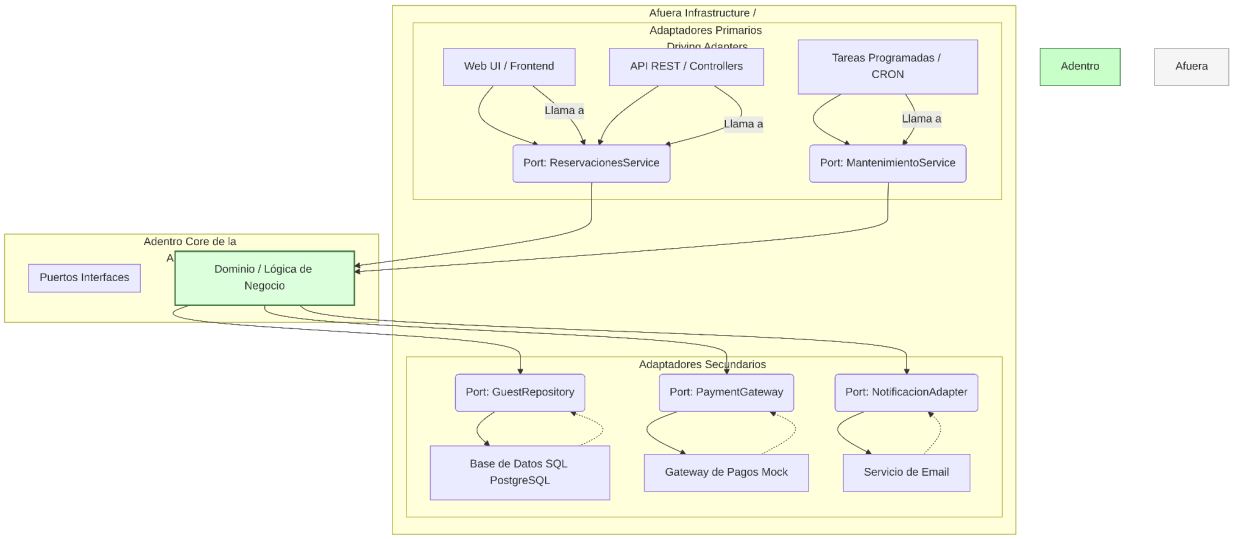
La adopción de la **Arquitectura Hexagonal** (también conocida como *Ports and Adapters*) es una decisión técnica clave para el proyecto. Su objetivo principal es asegurar el **desacoplamiento total** de la lógica de negocio central (el **dominio**) de las tecnologías de infraestructura, como la base de datos, los frameworks web y los servicios de mensajería.

Esta arquitectura se basa en el principio de la **Inversión de Dependencias**, donde el núcleo del sistema sólo define las interfaces que necesita (Puertos), y la infraestructura (Adaptadores) es la que se encarga de implementar esas interfaces. Esto garantiza tres beneficios estratégicos:

1. **Testabilidad Máxima:** Permite probar el dominio sin la necesidad de levantar la base de datos o el servidor, utilizando *mocks* para simular la infraestructura.
2. **Independencia Tecnológica:** Se puede cambiar la base de datos (por ejemplo, de PostgreSQL a MongoDB) o el framework de la API (de Node.js a Java Spring) sin modificar una sola línea de código de las reglas de negocio.
3. **Flexibilidad de Interfaz:** Facilita la adición de nuevas interfaces de usuario o entradas de datos (por ejemplo, una aplicación móvil o un sistema de voz) con mínimo esfuerzo.

Diagrama de la Arquitectura

A continuación, se muestra el diagrama que representa esta arquitectura para el sistema de gestión hotelera.



Componentes Clave del Dominio

El dominio es la parte agnóstica a la tecnología, dedicada exclusivamente a resolver los problemas del negocio hotelero.

- **Dominio / Lógica de Negocio (G):**
  - Contiene las Entidades de Dominio (Reservacion, Huesped, Habitacion), los Value Objects y las Reglas de Negocio.
  - Aquí reside la lógica crucial de "StayEasy", como la validación de disponibilidad de fechas, el cálculo de tarifas especiales, y la aplicación de políticas de cancelación.
  - El Dominio nunca importa o hace referencia a ninguna clase o librería de infraestructura.
- **Puertos Primarios (Driving Ports, P1, P2):**
  - Son las **interfaces de servicio** definidas por el Dominio que la aplicación expone. Definen lo que el sistema *puede hacer*.
  - Ejemplo (P1: ReservacionesService): Define operaciones como crearReservacion(), modificarEstadia() o procesarCheckOut().
- **Puertos Secundarios (Driven Ports, P3, P4, P5):**
  - Son las **interfaces de repositorio** o de servicio externo que el Dominio *necesita* para funcionar. Definen lo que el sistema *necesita del exterior*.
  - Ejemplo (P3: GuestRepository): Define los métodos para persistencia de datos (ej. encontrarPorId(), guardar()). El dominio llama a estos métodos sin saber si se están ejecutando contra SQL, NoSQL o un archivo local.

Componentes Clave los driven

Los driven es la capa que implementan los Puertos utilizando tecnologías concretas. Es

la única parte que depende de librerías o frameworks de terceros.

- **Adaptadores Primarios (Driving Adapters, A, B, C):**
  - Traducen eventos externos (solicitudes HTTP, clics, eventos programados) en llamadas al dominio a través de los Puertos Primarios.
  - Ejemplo (B: Adaptador API REST): Un Controller de Express/Spring recibe un JSON, lo valida, lo convierte a un objeto de Dominio (DTO/Mapper) y llama a ReservacionesService.crearReservacion().
- **Adaptadores Secundarios (Driven Adapters, D, E, F):**
  - Implementan los Puertos Secundarios, realizando la conexión y las operaciones específicas de la tecnología.
  - Ejemplo (D: Adaptador de Base de Datos PostgreSQL): Implementa la interfaz GuestRepository, traduciendo los objetos de Dominio a filas SQL y viceversa, gestionando la conexión a la base de datos PostgreSQL.
  - Ejemplo (E: Adaptador de Pagos Mock): Implementa la interfaz PaymentGateway para simular la comunicación con un servicio de pagos real, lo que permite al Dominio funcionar sin depender de la pasarela final.

Estrategia y Aplicación en "StayEasy"

La Arquitectura Hexagonal dicta un flujo de trabajo claro que minimiza los riesgos:

1. **Prioridad al Dominio:** La implementación comenzará con el diseño de las Entidades y Servicios de Dominio (G) y la definición de todos los Puertos (P1-P5). Esto se hará antes de escribir la primera línea de código de la interfaz de usuario o de la base de datos.
2. Desarrollo Paralelo: Una vez definidos los Puertos (interfaces), los equipos de Frontend (Adaptadores Primarios) y Backend/DB (Adaptadores Secundarios) podrán trabajar en paralelo con alta confianza. El desarrollador Backend puede implementar el Adaptador de Base de Datos (D) mientras el equipo de QA puede probar la lógica de ReservacionesService (G) utilizando un Adaptador Secundario falso (mock) en lugar de la base de datos real.
3. **Transparencia de Datos:** La transformación de los datos desde la Base de Datos (SQL) al Dominio (Objetos) y viceversa es manejada exclusivamente por los Adaptadores Secundarios, asegurando que el Dominio nunca "conozca" la estructura de las tablas SQL.

Esta estructura provee el marco sólido y profesional necesario para construir un Sistema de Gestión Hotelera robusto y listo para el crecimiento futuro.

Análisis de riesgos

No. Riesgo	Riesgo	Impacto Potencial	Probabilidad de que ocurra	Nivel de Riesgo
ARN-001	Falla en el servidor de producción	Interrupción del servicio, pérdida temporal de acceso al sistema	Media	CRITICO
ARN-002	Pérdida de la base de datos por falta de respaldo	Pérdida de información de huéspedes y reservaciones	Baja	CRITICO
ARN-003	Error en el módulo de reservaciones	Doble asignación de habitaciones o overbooking	Alta	CRITICO
ARN-004	Falla en la autenticación de usuarios	Acceso no autorizado a datos sensibles	Media	ALTO
ARN-005	Caída del servicio de internet en el hotel	Imposibilidad de usar el sistema en línea	Alta	MEDIO
ARN-006	Vulnerabilidad de seguridad no detectada	Robo de datos personales de huéspedes	Media	CRITICO
ARN-007	Retraso en el desarrollo del backend	Atraso en la entrega del proyecto final	Media	MEDIO
ARN-008	Errores de integración entre frontend y backend	Fallas en la comunicación API (Ktor–React)	Media	ALTO
ARN-009	Saturación del servidor por exceso de usuarios concurrentes	Lentitud o caídas del sistema	Baja	ALTO

ARN-010	Falta de capacitación del recepcionista	Uso incorrecto del sistema y errores operativos	Alta	MEDIO
ARN-011	Falla en la conexión a la base de datos PostgreSQL	Imposibilidad de guardar o consultar datos	Media	CRITICO
ARN-012	Incompatibilidad del sistema con navegadores no soportados	Fallos visuales o funcionales en la interfaz	Media	MEDIO
ARN-013	Error en la generación de facturas	Cálculos erróneos o problemas fiscales	Media	ALTO
ARN-014	Pérdida de acceso al repositorio GitHub	Imposibilidad temporal de colaboración	Baja	MEDIO
ARN-015	Corrupción de datos en migraciones	Inconsistencias entre reservas y facturas	Baja	ALTO
ARN-016	Fallo en pruebas unitarias o E2E	Defectos no detectados antes del despliegue	Media	ALTO
ARN-017	Error de configuración del entorno de despliegue (CI/CD)	Fallo en la publicación o actualización del sistema	Media	MEDIO
ARN-018	Conflicto de versiones de dependencias	Fallas de compilación o incompatibilidades en módulos	Media	MEDIO
ARN-019	Baja disponibilidad del sistema	Pérdida de confianza del cliente	Baja	ALTO

	(menos del 99.5%)			
ARN-020	Ataque DDoS o intento de hackeo	Interrupción total del servicio hotelero	Baja	CRITICO

Recopilacion de informacion (Formulacion)

1. ¿Cuál es su métrica operativa clave diaria para el éxito inmediato (aparte de la ocupación)?
2. ¿Cómo se asegura la eficiencia en el uso de los recursos del hotel (costos operativos)?
3. Describa su proceso para elaborar y controlar el presupuesto anual.
4. ¿Cuál es su estrategia para minimizar los gastos variables sin afectar la calidad del servicio?
5. ¿Cómo maneja la gestión de inventario y compras para maximizar los márgenes?
6. ¿Cuál es la importancia del RevPAR (Ingreso por Habitación Disponible) en su estrategia a corto plazo?
7. ¿Cómo monitorea y mejora el desempeño del F&B (Alimentos y Bebidas)?
8. ¿Qué sistemas de PMS (Property Management System) ha utilizado y cuál prefiere?
9. Describa un momento en que tuvo que reducir costos drásticamente. ¿Qué acciones tomó?
10. ¿Cómo gestiona el mantenimiento preventivo y correctivo de las instalaciones?
11. ¿Cuál es su enfoque para asegurar el cumplimiento de normativas de salud y seguridad?
12. ¿Cómo integrar la sostenibilidad en las operaciones diarias para reducir la huella de carbono?
13. ¿Qué estrategias implementa para asegurar que los departamentos trabajen sinérgicamente?
14. ¿Cómo audita la calidad y limpieza en las habitaciones de manera consistente?
15. ¿Qué haría si la puntuación de satisfacción de limpieza cayera por debajo del estándar?
16. Describa brevemente su filosofía de liderazgo.
17. ¿Cómo se promueve una cultura de excelencia y servicio en todos los niveles del personal?
18. ¿Cuál es su estrategia principal para reducir la alta rotación de personal en la industria?
19. Describa un programa de capacitación que haya implementado exitosamente.
20. ¿Cómo evalúa el rendimiento individual de los jefes de departamento?
21. ¿Cómo manejar un conflicto significativo entre dos jefes de departamento?
22. ¿Qué programas ha implementado para el desarrollo de futuros líderes internos?
23. ¿Cómo mantiene alta la moral del equipo en temporadas bajas o bajo estrés?
24. ¿Qué importancia le da a la diversidad e inclusión en la contratación y gestión?
25. ¿Cómo gestiona las expectativas salariales de los empleados con el presupuesto?
26. ¿Cuál es su enfoque para dar feedback constructivo y correctivo a un empleado clave?
27. ¿Cómo delegar responsabilidades de manera efectiva sin perder el control de la calidad?
28. ¿Cómo asegura que la comunicación fluya de la gerencia a la primera línea y viceversa?
29. Cuénteme sobre un error que cometió como líder y qué aprendió de él.
30. ¿Qué pasos toma para asegurar un ambiente de trabajo libre de acoso o discriminación?

Stack Tecnológico

La selección del *Stack* se realiza con base en el cumplimiento de los principios de la Arquitectura Hexagonal, buscando la **seguridad de tipos**, la **escalabilidad** y la **eficiencia** en el desarrollo.

Resumen del Stack Tecnológico Seleccionado

Rol en el Sistema	Componente del Modelo Hexagonal	Tecnología	Justificación Clave
Backend / Core	Dominio (G)	**Kotlin**	Lenguaje moderno, seguro contra nulos (Null-Safety), y altamente legible, perfecto para modelar el Core de Negocio.
API REST	Adaptador Primario (B)	**Ktor**	Framework ligero y asíncrono, nativo de Kotlin, ideal para ser un Adaptador *delgado* que solo expone el Dominio sin añadir sobrecarga.
Persistencia	Adaptador Secundario (D)	**PostgreSQL**	Base de datos relacional robusta, conocida por su fiabilidad en entornos transaccionales de alta concurrencia (ACID).
Frontend / Web UI	Adaptador Primario (A)	**React + TypeScript**	Proporciona una interfaz dinámica, modular y reactiva, esencial para una recepción de hotel eficiente. TypeScript añade tipado seguro de extremo a extremo (Frontend a Backend).

Justificación Técnica por Componente

Backend y Core: Kotlin y Ktor

- **Kotlin (Core/Dominio):** Al ser un lenguaje de la JVM, nos proporciona la madurez del ecosistema, mientras que su sintaxis concisa y su tipado estricto (Null-Safety) reducen la probabilidad de errores en la lógica de negocio crítica (reservaciones, tarifas). El Dominio será escrito en Kotlin puro.
- **Ktor (Adaptador API REST):** Es la mejor opción para la arquitectura Hexagonal. Su diseño minimalista y su uso nativo de **Coroutines** nos permitirá crear un Adaptador API REST (B) que sea simplemente un **puente asíncrono** entre el Frontend (Adaptador A) y el Dominio (G). Esto garantiza que el Core no esté contaminado por el *framework* web, manteniendo la limpieza y la testabilidad.

Persistencia: PostgreSQL

- **PostgreSQL (Adaptador DB):** Se elige por su fiabilidad transaccional, su soporte robusto para concurrencia (especialmente relevante para la gestión de habitaciones) y sus funcionalidades avanzadas (como JSONB para campos flexibles y excelente manejo de índices geográficos si se requiere crecimiento futuro). Para la implementación, se recomienda usar una librería ORM ligera de Kotlin como **Exposed** o **Ktorm** dentro del Adaptador de Base de Datos (D).

Frontend: React y TypeScript

- **React (Adaptador Web UI):** Es el estándar de la industria para construir **Single Page Applications (SPA)**. Su enfoque basado en componentes es perfecto para un sistema modular como "StayEasy", permitiendo que Esteban (Desarrollador Front End) construya rápidamente módulos reutilizables para el *Dashboard*, el *Calendario de Reservas* y el *Formulario de Check-in*.
- **TypeScript (Control de Calidad):** Se implementará TypeScript en todo el Frontend. Esto es crucial porque permite definir interfaces de datos que **coincidan exactamente** con los DTOs (Data Transfer Objects) que envía la API de Ktor. Esto crea una **cadena de tipado seguro de extremo a extremo** (PostgreSQL → Kotlin/Ktor → React/TypeScript), eliminando la mayoría de los errores relacionados con tipos que suelen aparecer en la fase de integración.

Herramientas de Desarrollo y Flujo de Trabajo

Para optimizar la colaboración y el flujo de trabajo:

- **Gestión de Dependencias: Gradle Kotlin DSL** se utilizará en el Backend para una gestión de dependencias tipada. **npm/Yarn** se usarán en el Frontend.
- **Control de Versiones: Git** y un servicio de repositorio (GitHub/GitLab) serán obligatorios para el trabajo en paralelo.

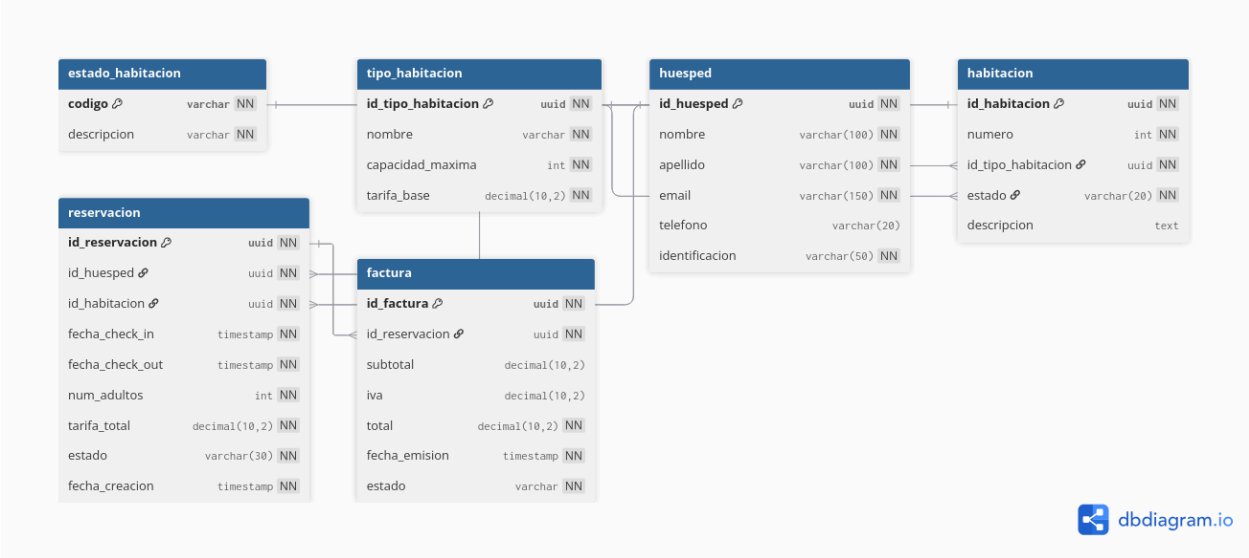
- **IDE: IntelliJ IDEA** es altamente recomendado para todo el equipo debido a su soporte nativo y avanzado para Kotlin, Ktor y React/TypeScript.

Diseño de Base de Datos (PostgreSQL)

El diseño de la Base de Datos Relacional (PostgreSQL) se enfocará en la **normalización** y la **integridad transaccional (ACID)** para manejar de manera robusta los datos críticos del hotel, como la disponibilidad de habitaciones y las transacciones de pago. Este diseño soporta las Entidades de Dominio definidas en el Core de la aplicación.

Diagrama Relacional

El siguiente diagrama representa las entidades centrales del sistema y sus relaciones, enfocándose en el módulo de reservaciones y gestión de habitaciones.



Justificación de la Persistencia

El sistema "StayEasy" requiere una solución de persistencia que garantice la **Integridad Transaccional (ACID)** y la fiabilidad de los datos, dado que se manejan procesos financieros y de disponibilidad críticos (reservaciones, tarifas, facturación). Por ello, se selecciona **PostgreSQL** como Sistema de Gestión de Bases de Datos Relacional (SGBDR).

Ventajas Clave de PostgreSQL:

1. **Robustez Transaccional:** Su madurez garantiza que las operaciones como la reservación de una habitación sean atómicas; es decir, que se completen completamente o no se realicen en absoluto, evitando el *overbooking* por fallos concurrentes.
2. **Escalabilidad y Concurrencia:** PostgreSQL maneja eficientemente las transacciones concurrentes (ej. múltiples recepcionistas en Check-in) sin bloqueos excesivos, crucial para un hotel en operación constante.
3. **Seguridad de Tipos:** Su tipado estricto se alinea con la seguridad que buscamos en el

Backend con Kotlin/Ktor.

Estructura y Normalización

El diseño se basa en el modelo relacional, buscando la Tercera Forma Normal (3NF) para minimizar la redundancia y optimizar las consultas a través de claves primarias (uuid) y claves foráneas.

Entidades Principales del Dominio

Entidad	Propósito	Clave Primaria
huesped	Almacena la información de los clientes. El campo identificacion (INE, Pasaporte) es crucial para el Check-in y se define como único.	id_huesped (uuid)
habitacion	Registra cada unidad física del hotel, incluyendo su número único y su relación con un tipo_habitacion.	id_habitacion (uuid)
reservacion	El núcleo del sistema. Almacena las fechas de Check-in y Check-out, la relación con el huesped y la habitacion, y el estado actual del servicio.	id_reservacion (uuid)
factura	Almacena los detalles financieros y fiscales asociados a una única reservacion. Su diseño asegura una relación uno a uno (1:1) con la reservación.	id_factura (uuid)

Catálogos y Control

Catálogo	Propósito	Campo Clave Foránea
tipo_habitacion	Catálogo fijo para definir las características y tarifas de las habitaciones (ej. Suite, Doble, King).	id_tipo_habitacion en habitacion
estado_habitacion	Catálogo de estados físicos de una habitación (ej. LIMPIA, SUCIA, MANTENIMIENTO).	estado en habitacion

Detalle del Esquema de Tablas (DDL Lógico)

A continuación, se detalla la estructura lógica de las tablas, incluyendo las columnas, tipos de datos y restricciones de integridad:

Tabla huesped

Columna	Tipo de Dato	Restricciones	Propósito
id_huesped	UUID	PK, NOT NULL	Identificador único del huésped.
nombre	VARCHAR(100)	NOT NULL	Nombre del huésped.
apellido	VARCHAR(100)	NOT NULL	Apellido del huésped.
email	VARCHAR(150)	NOT NULL, UNIQUE	Correo electrónico, usado para notificaciones.
telefono	VARCHAR(20)	N/A	Teléfono de contacto.
identificacion	VARCHAR(50)	NOT NULL, UNIQUE	Identificador oficial (INE, Pasaporte) para Check-in.

Tabla tipo_habitacion (Catálogo)

Columna	Tipo de Dato	Restricciones	Propósito
id_tipo_habitacion	UUID	PK, NOT NULL	Identificador único del tipo de habitación.
nombre	VARCHAR(50)	NOT NULL	Nombre descriptivo (ej. "Suite",

			"Doble", "King").
capacidad_maxima	INTEGER	NOT NULL	Número máximo de personas permitidas.
tarifa_base	DECIMAL(10, 2)	NOT NULL	Precio por noche para este tipo, antes de descuentos.

Tabla estado_habitacion (Catálogo)

Columna	Tipo de Dato	Restricciones	Propósito
codigo	VARCHAR(20)	PK, NOT NULL	Código abreviado (ej. Limpia, Sucia, Mantenimiento).
descripcion	VARCHAR(100)	NOT NULL	Descripción completa del estado.

Tabla habitacion

Columna	Tipo de Dato	Restricciones	Propósito
id_habitacion	UUID	PK, NOT NULL	Identificador único de la habitación.
numero	INTEGER	NOT NULL, UNIQUE	Número físico asignado (ej. 101, 205).
id_tipo_habitacion	UUID	FK, NOT NULL	Relación con la tabla tipo_habitacion.

estado	VARCHAR(20)	FK, NOT NULL	Estado actual (limpieza/ocupación). Relación con estado_habitacion.
descripcion	TEXT	N/A	Breve descripción de la habitación o notas.

Tabla reservacion

Columna	Tipo de Dato	Restricciones	Propósito
id_reservacion	UUID	PK, NOT NULL	Identificador único de la reserva.
id_huesped	UUID	FK, NOT NULL	Huésped principal que realiza la reserva.
id_habitacion	UUID	FK, NOT NULL	Habitación reservada.
fecha_check_in	TIMESTAMP	NOT NULL	Fecha y hora de entrada esperada/real.
fecha_check_out	TIMESTAMP	NOT NULL	Fecha y hora de salida esperada/real.
num_adultos	INTEGER	NOT NULL, DEFAULT: 1	Número de adultos en la reserva.
tarifa_total	DECIMAL(10, 2)	NOT NULL	Costo final de la reserva.

estado	VARCHAR(30)	NOT NULL	Estado del ciclo de vida (CONFIRMADA, CHECK_IN, CANCELADA).
fecha_creacion	TIMESTAMP	NOT NULL	Momento en que se registró la reserva.

Tabla factura

Columna	**Tipo de Dato**	**Restricciones**	**Propósito**
id_factura	UUID	PK, NOT NULL	Identificador único de la factura.
id_reservacion	UUID	FK, NOT NULL, UNIQUE	Clave foránea única, asegura 1:1 con reservacion.
subtotal	DECIMAL(10, 2)	N/A	Monto antes de impuestos.
iva	DECIMAL(10, 2)	N/A	Monto del impuesto.
total	DECIMAL(10, 2)	NOT NULL	Monto total final.
fecha_emision	TIMESTAMP	NOT NULL	Fecha de creación de la factura.
estado	VARCHAR(20)	NOT NULL	Estado del pago (Pagada, Pendiente, Anulada).

Modelo de Relaciones

Las relaciones garantizan la integridad referencial:

- **Relación Huésped-Reservación (1:N):** Un huésped puede realizar múltiples reservaciones.
- **Relación Habitación-Reservación (1:N):** Una habitación puede tener múltiples reservaciones a lo largo del tiempo.
- Relación Reservación-Factura (1:1): Cada reservación genera una única factura, aplicando la restricción UNIQUE en la FK de la tabla factura.

Este diseño proporciona la base de datos necesaria para soportar la lógica de negocio del Dominio (Core) de manera desacoplada, ya que la comunicación con PostgreSQL se gestionará a través del **Adaptador Secundario (D)**, manteniendo la pureza del Hexagonal.

Estrategia de Pruebas y Aseguramiento de Calidad (QA)

El Aseguramiento de Calidad (QA) es un proceso continuo que garantiza que el sistema de gestión hotelera "StayEasy" no solo cumpla con los requerimientos funcionales, sino que también sea robusto, seguro y escalable. **Ian** liderará la definición de casos de prueba y la coordinación de las pruebas de aceptación.

La estrategia se dividirá en tres niveles principales, que se alinean perfectamente con la Arquitectura Hexagonal (Sección 3), utilizando la **seguridad de tipos** provista por Kotlin y TypeScript.

Pruebas Unitarias (Nivel Core y Adaptadores)

Estas pruebas se enfocan en verificar la **lógica individual** de las funciones y los componentes, asegurando que el **Dominio (G)** y los **Adaptadores** funcionen correctamente de forma aislada.

- **Responsabilidad:** Principalmente **Ángel (Backend)** y **Esteban (Frontend)**.
- **Enfoque en el Backend (Kotlin/Ktor):**
    - **Herramientas:** Se utilizará **JUnit 5** como framework de pruebas principal. Para el aislamiento y la simulación de dependencias, se empleará **MockK**, la librería de *mocking* nativa e idiomática de Kotlin.
    - Core del Dominio (Purity Test): Se probarán los Servicios y las Entidades del Core (G) utilizando MockK para simular las llamadas a los Puertos. Por ejemplo, se mockeará el ReservacionRepository para validar la lógica de negocio pura (ej. la función calcularTarifa o validarDisponibilidad) sin tocar la base de datos.
    - Adaptadores: Se probará la correcta implementación de los Adaptadores de Ktor (B), asegurando que el routing HTTP se mapee correctamente a las funciones de los Servicios del Dominio.
- **Enfoque en el Frontend (React/TypeScript):**
    - Herramientas: Se utilizarán Jest (runner) y React Testing Library (RTL). RTL se prefiere para enfocarse en el comportamiento del usuario (user-centric testing).
    - Componentes: Se validará que los componentes clave (ej. CalendarComponent o CheckInForm) se rendericen correctamente y que respondan a los eventos del usuario (clicks, entradas de datos) según lo esperado.

Pruebas de Integración y End-to-End (Nivel Conexión)

Las pruebas de integración validan que los diferentes componentes del *Stack* interactúen correctamente, asegurando la fluidez de datos a través de los **Puertos**.

- **Responsabilidad: Ángel (Backend)** en coordinación con **Esteban (Frontend)** e **Ian (QA)**.
- **Integración del Backend (Adaptador DB):**
    - Se ejecutará un subconjunto de pruebas de integración contra una instancia de **PostgreSQL ejecutándose en Docker** o un servicio en memoria (como H2, aunque con precaución). Esto validará que el Adaptador Secundario (D)

(usando Exposed/Ktorm) interactúe correctamente con el esquema de la base de datos (Sección 5).
    - **Focus:** Garantizar la integridad transaccional (ACID) en operaciones críticas como la creación simultánea de reservas.
- **Pruebas End-to-End (E2E) con Cypress:**
    - **Herramienta:** Se utilizará **Cypress** para simular el flujo completo del usuario en el navegador.
    - **Alcance:** Probar el flujo desde la interfaz de React (Adaptador A), pasando por la API de Ktor (Adaptador B), y confirmando la persistencia de los datos en PostgreSQL (Adaptador D). Esto valida la **Cadena de Tipado Seguro** de extremo a extremo.

Pruebas de Aceptación del Usuario (UAT)

Las UAT verifican la usabilidad y la alineación con los requerimientos de negocio, documentando la aceptación formal de las características.

- **Responsabilidad:** Principalmente **Ian (QA)** y **Fernando (Documentación)**.
- **Ejecutores:** Personal que simule al usuario final (recepcionistas/administradores).
- Documentación: Ian definirá los escenarios detallados de prueba en una matriz formal (Matriz de UAT), que incluirá: ID del Caso, Precondiciones, Pasos, Resultado Esperado y Criterio de Aceptación. El resultado final debe ser un documento de Aprobación de UAT (Sign-off).
- **Caso Crítico de Concurrencia:** Probar la **gestión de la concurrencia** intentando que dos "usuarios" realicen una reserva para la misma habitación al mismo tiempo. El sistema debe gestionar el bloqueo de manera elegante y solo permitir una reserva, validando la lógica transaccional de Kotlin.

Pruebas de Despliegue e Infraestructura (Administradas por Uriel)

Estas pruebas aseguran que el sistema sea estable, escalable y seguro en su entorno de ejecución.

- **Pruebas de Carga y Estrés:**
    - **Herramienta:** Se usará **Gatling** (o JMeter) para simular escenarios de alta demanda (ej. 100 usuarios concurrentes intentando consultar habitaciones).
    - **Métrica:** Se establecerán **SLAs (Acuerdos de Nivel de Servicio)** que dictaminen, por ejemplo, que el 95% de las peticiones de reserva deben responder en menos de 500 ms.
- **Integración Continua / Despliegue Continuo (CI/CD):**
    - **Herramienta:** Se configurará un *pipeline* en **GitLab CI** o **GitHub Actions** que, en cada *push* a la rama principal, ejecute automáticamente todas las pruebas Unitarias y de Integración antes de permitir el despliegue a los entornos de *staging* o producción.

Metodología Ágil y Proceso

Para gestionar el desarrollo del sistema "StayEasy" y fomentar la colaboración efectiva, se adoptará un enfoque **Híbrido Ágil** que combina la estructura de **Scrum** con la gestión de flujo de **Kanban**. Esta metodología está diseñada para un equipo universitario, maximizando la visibilidad, la comunicación y la adaptación al cambio.

Principios y Valores Fundamentales

El equipo "StayEasy" se regirá por los siguientes principios ágiles adaptados al contexto universitario:

- **Entrega Temprana y Frecuente:** Entregar funcionalidades operativas al final de cada Sprint de dos semanas para obtener retroalimentación rápida.
- **Colaboración Constante:** Fomentar la comunicación directa y simple, especialmente entre Frontend (Esteban) y Backend (Ángel) sobre las interfaces de la API.
- **Simplicidad:** Enfocarse en la funcionalidad requerida por el MVP, evitando complejidades innecesarias en el diseño y el código.
- **Adaptación al Cambio:** Priorizar la flexibilidad sobre seguir un plan rígido; si Ian (QA) identifica una nueva necesidad, el *backlog* se ajusta en la siguiente planificación.
- **Conocimiento Compartido:** Documentar bien el código (Fernando) y el diseño (Daniel) para mitigar el riesgo de dependencia de un solo experto.

Justificación de la Elección: Scrum Híbrido

Se elige Scrum/Kanban ligero por las siguientes razones:

1. **Ritmo y Foco (Scrum):** Los Sprints de dos semanas proporcionan un ritmo predecible y un objetivo claro que ayuda al equipo a mantener la concentración en tareas específicas.
2. **Gestión de Flujo (Kanban):** El tablero visual es clave para **Uriel (Infraestructura)**, **Ian (QA)** y **Daniel (Jefe de Proyecto)**, permitiendo ver cuellos de botella (ej. demasiadas tareas en el estado "En Revisión") y equilibrar la carga de trabajo.
3. **Alineación Universitaria:** La estructura de eventos (Standups diarios) mantiene al equipo cohesionado sin el rigor extremo de Scrum corporativo.

Roles y Responsabilidades Metodológicas

Aunque los roles funcionales (Desarrollador, QA) se mantienen, se complementan con responsabilidades metodológicas:

Rol	Responsable Principal	Responsabilidad Metodológica

Jefe de Proyecto	Daniel	Lidera las reuniones de Planificación y Retrospectiva. Mantiene el *Backlog* priorizado.
Equipo de Desarrollo	Ángel, Esteban, Uriel	Ejecuta las tareas del Sprint, estima el esfuerzo de las tareas y se adhiere a la **Definición de Hecho (DoD)**.
Garante de Calidad (QA)	Ian	Responsable del estado "En Revisión". Asegura que las tareas cumplen con los criterios de aceptación y la DoD antes de pasar a "Terminado".
Gestión de Documentación	Fernando	Documenta las decisiones y prepara los entregables del Sprint (ej. manuales de usuario para las funcionalidades terminadas).

Ciclo de Vida y Fases (Conexión al Cronograma)

El ciclo de vida del desarrollo se divide en Sprints de dos semanas, siguiendo el flujo del cronograma de 11 semanas (Sección 7):

- **Sprints 1-2 (Fase 1 y 2 Inicio):** Enfoque en la **Arquitectura y Core (Dominio G)**. Se definen las interfaces del Core y se implementan las Entidades.
- **Sprints 3-5 (Fase 2 y 3):** Desarrollo de **Adaptadores (B y A)**. Integración inicial de Ktor con React/TS. El *criterio de terminación* es la API funcional.
- **Sprints 6-8 (Fase 4 y 5):** Estabilización, Pruebas y Cierre. Se ejecutan las pruebas E2E, UAT, y se realiza la corrección intensiva de errores (*Debugging*).

Gestión de Tareas (Tablero Kanban)

Se utilizará una herramienta digital (ej. Trello, Asana o Jira) para gestionar el flujo de trabajo a través del tablero Kanban, asegurando que solo haya un número limitado de tareas en la columna "En Progreso" para evitar el *multitasking*.

Estado del Tablero	Flujo de Tareas	Proceso de Verificación

To Do	Priorización semanal en la Planificación del Sprint.	Decidida por Daniel y el equipo.
In Progress	Tarea que el Desarrollador (Ángel/Esteban) está codificando.	**El desarrollador no puede tener más de 2 tareas aquí.**
In Review / QA	La tarea está codificada y pasa a Ian (QA) para verificación.	**Ian ejecuta los casos de prueba (Sección 6)**.
Done	La tarea ha superado las pruebas de Ian y cumple con la DoD.	Lista para fusionar a la rama principal (*main*).

Artefactos y Entregables

Los siguientes artefactos se generarán y mantendrán actualizados durante el ciclo de vida:

1. **Product Backlog:** Lista priorizada de funcionalidades requeridas para "StayEasy".
2. **Sprint Backlog:** Subconjunto de tareas seleccionadas del *Backlog* para el Sprint actual.
3. **Código Funcional:** El software de trabajo que se despliega al entorno de QA al final de cada Sprint.
4. **Documentación Técnica:** Manuales de API (Swagger), diagrama de base de datos (DBML) y documentación del Core (Fernando).

Herramientas CASE y Colaboración

Se utilizará una suite de herramientas sencillas y gratuitas para facilitar el proceso ágil:

Herramienta	Propósito	Responsable Principal
GitHub	Control de versiones y Repositorio Único.	Uriel (Infraestructura)
Plane.so	Tablero visual para la Gestión de Tareas	Daniel (Jefe de

		Proyecto)
Swagger / OpenAPI	Documentación automática del Adaptador API (B).	Ángel (Backend)
Discord	Comunicación Diaria.	Todos

Definición de Hecho (Definition of Done - DoD)

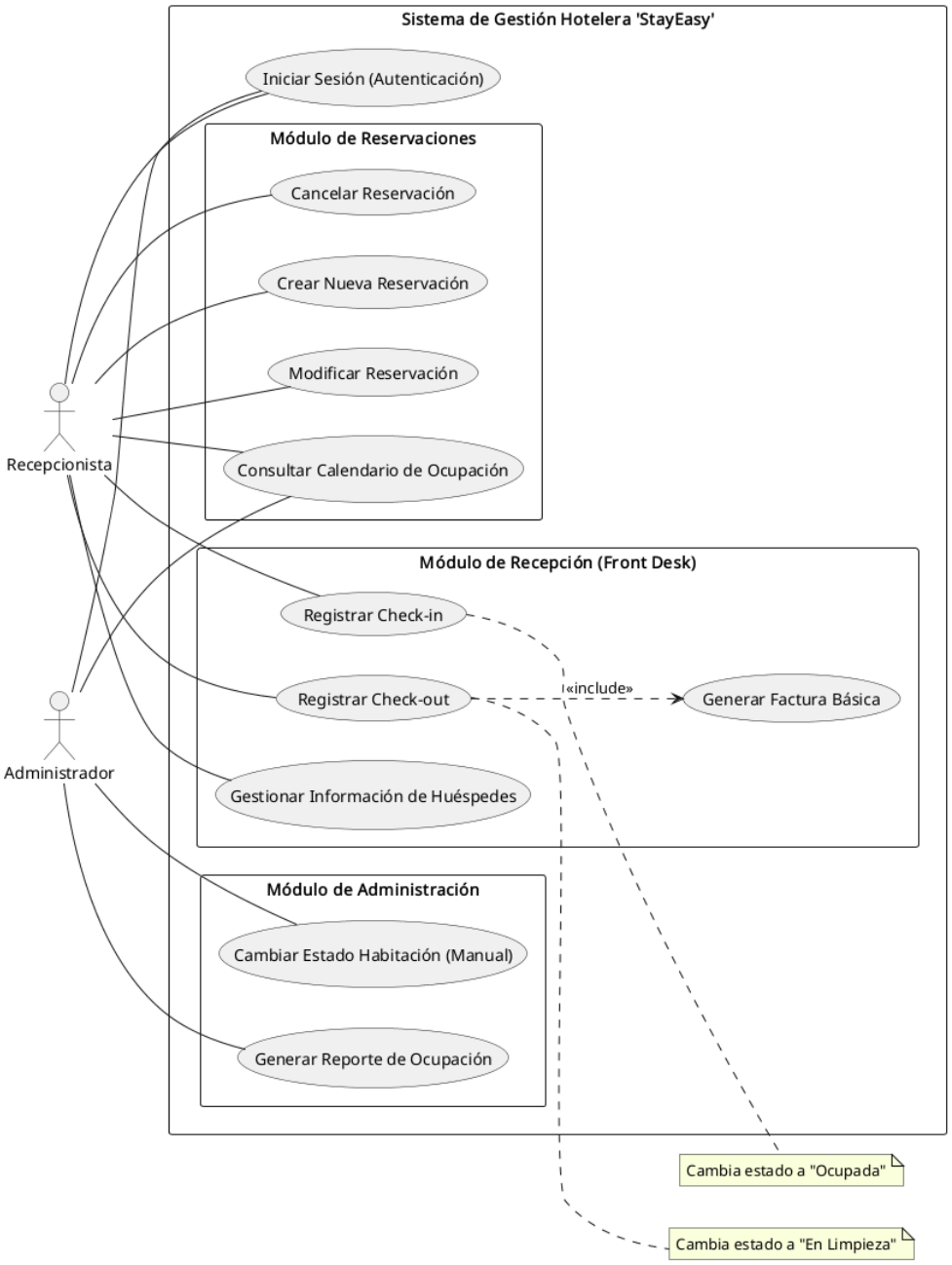
Una tarea solo se considera "Terminada" y puede pasar a la columna Done del Kanban si se cumplen TODOS los siguientes criterios:

- El código fuente está **documentado** (comentarios JSDoc/KDoc).
- Se han creado y pasado el **100% de las Pruebas Unitarias** para el Core (Sección 6).
- Se han creado y pasado las **Pruebas de Integración/E2E** relevantes (si la tarea es un Adaptador).
- El código ha pasado la revisión del par (Peer Review).
- El código está fusionado a la rama principal del repositorio (main).
- La funcionalidad cumple con los **Criterios de Aceptación** definidos en la historia de usuario.

Diseño Inicial / Bocetos

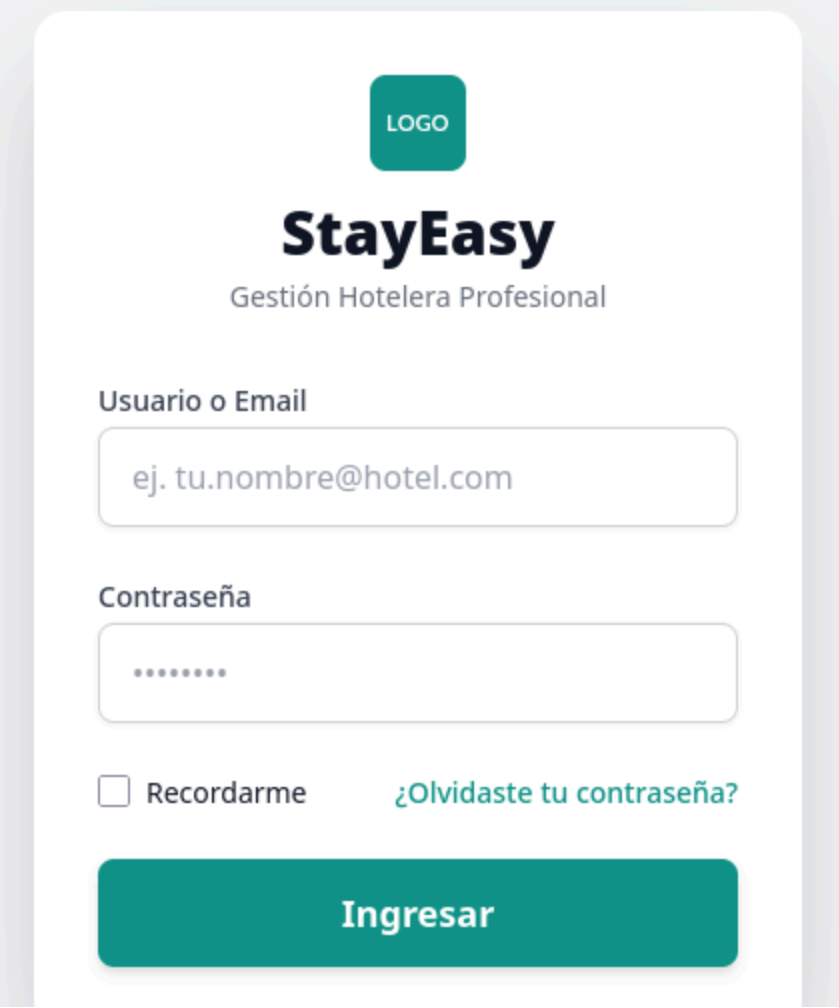
Diagrama de Casos de Uso Preliminar

A continuación, se presenta un diagrama de casos de uso que describe las interacciones de los actores principales (Recepcionista y Administrador) con los módulos del sistema

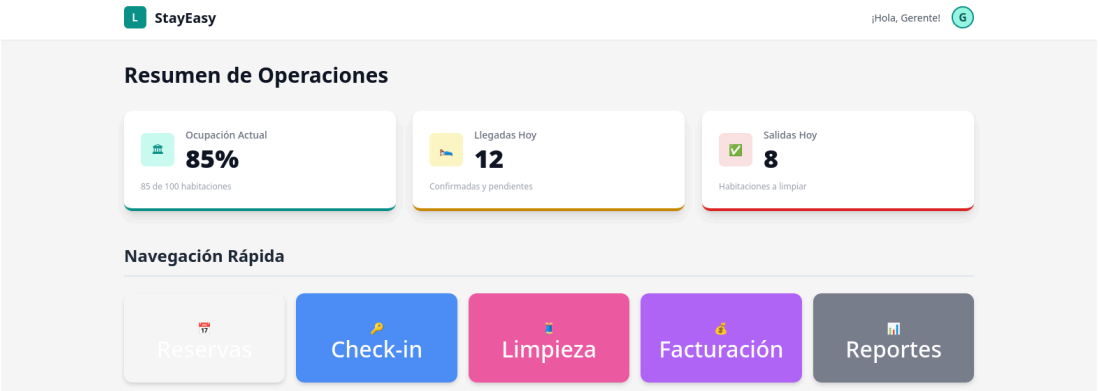


Bocetos / Prototipos Conceptuales

- **Pantalla de Inicio de Sesión:** Una interfaz limpia con campos para usuario, contraseña y un botón de "Ingresar".

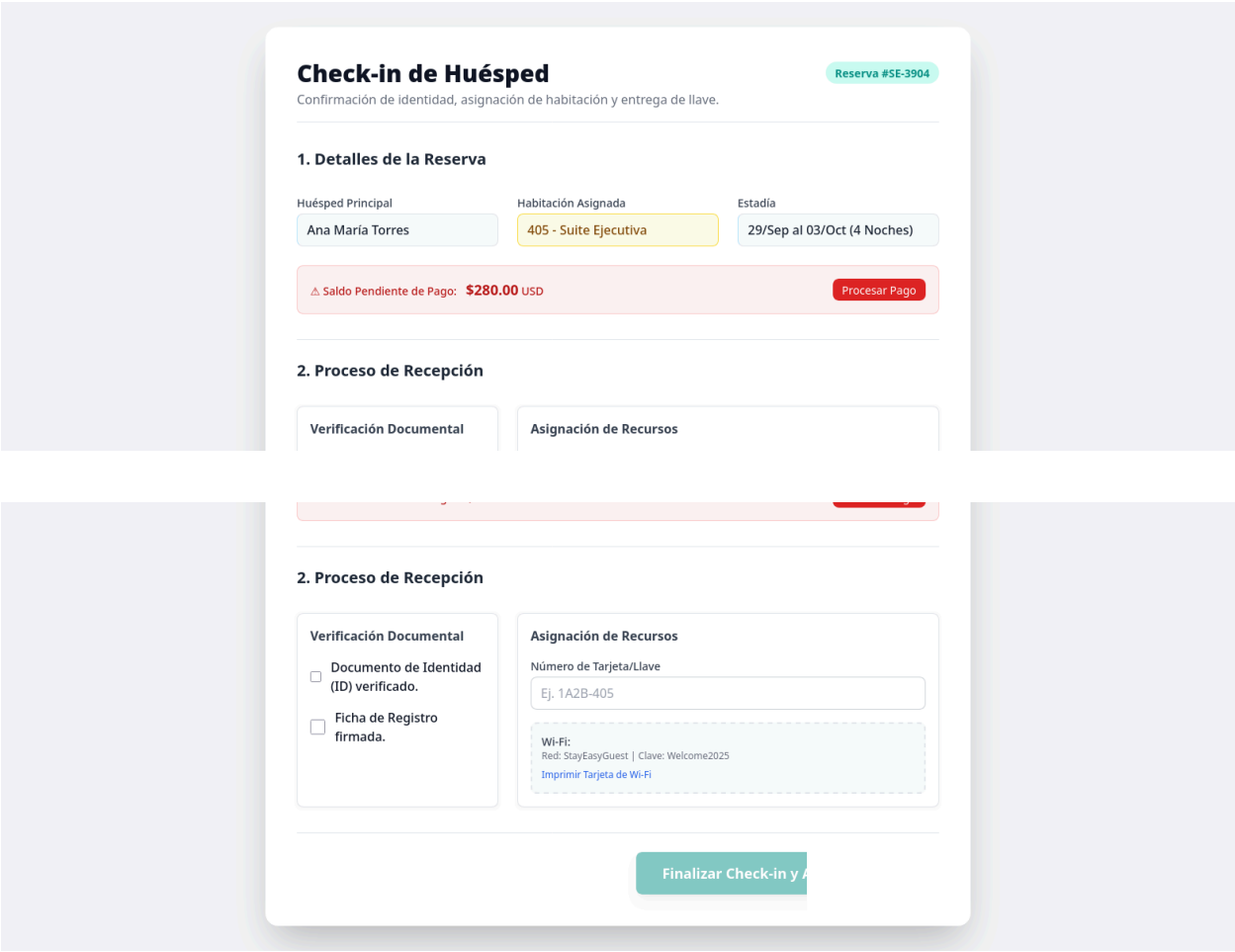


Dashboard Principal: Al iniciar sesión, el usuario verá un panel con un resumen del estado actual del hotel: número de llegadas esperadas, número de salidas, porcentaje de ocupación actual y un acceso rápido a las funciones principales.



- **Pantalla de Reservaciones:** Un calendario mensual o semanal que mostrará las habitaciones en el eje Y y los días en el eje X. Las celdas estarán coloreadas según el estado de la reserva. Al hacer clic en una fecha y habitación, se abrirá un formulario para crear una nueva reservación.

- **Formulario de Check-in:** Un formulario pre-llenado con los datos de la reserva, donde el recepcionista confirmará la identidad del huésped, asignará la llave y finalizará el proceso.

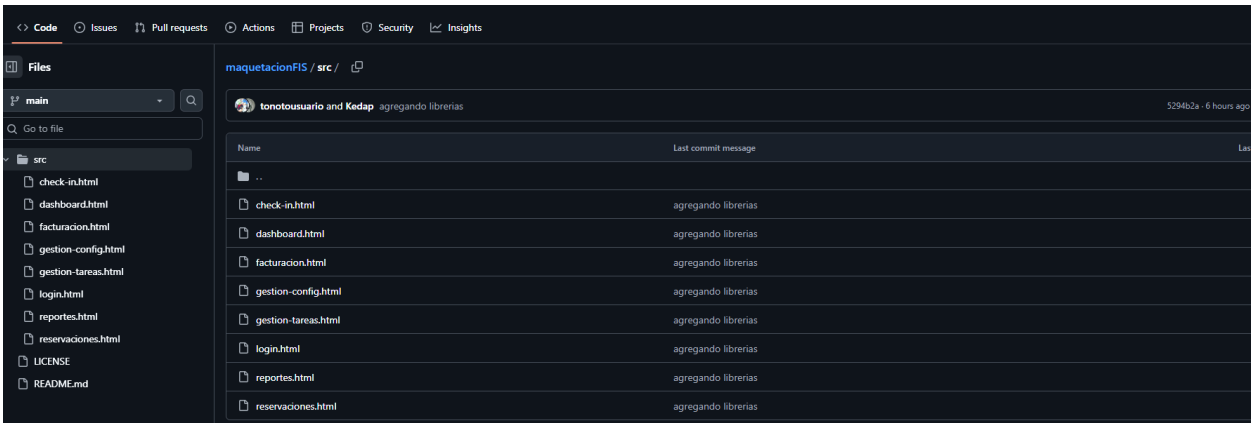


- **Maquetado:**

Nota sobre el código fuente:
Las capturas incluidas en este documento muestran el resultado visual (frontend) del proyecto.
El **código fuente completo** no se encuentra dentro de este archivo, ya que está alojado en mi repositorio de GitHub por motivos de organización y peso.

Puede acceder al código directamente en el siguiente enlace:
🔗 https://github.com/tonotousuario/maquetacionFIS

Dentro del repositorio, el código se encuentra en la carpeta `src`, que contiene los archivos HTML, CSS y JavaScript utilizados para la maquetación del sitio.



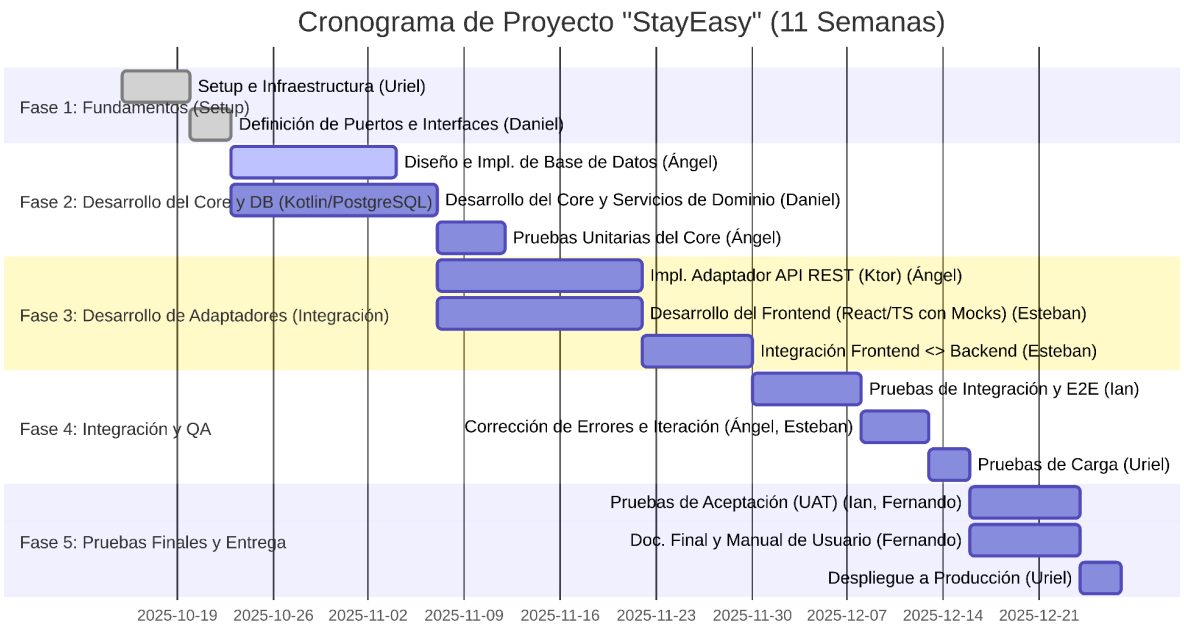
Planificación y Organización del Equipo

Roles Asignados

Miembro	Rol	Responsabilidades Principales
Daniel	Jefe de Proyecto y Arquitecto de Software	Liderar al equipo, tomar decisiones técnicas clave, definir la arquitectura del sistema.
Ángel	Desarrollador Backend y DBA	Implementar la lógica del servidor, APIs y gestionar el diseño y mantenimiento de la BD.
Esteban	Desarrollador Front End y Diseñador UI/UX	Diseñar y construir la interfaz de usuario, asegurando una buena experiencia de usuario.
Ian	Analista de Requerimientos y QA	Recopilar y documentar requerimientos, diseñar y ejecutar casos de prueba, asegurar calidad.
Fernando	Documentación y Soporte	Crear toda la documentación técnica y de usuario, preparar material de soporte.
Uriel	Administrador de Infraestructura	Configurar entornos y servidores, implementar CI/CD, supervisar seguridad básica y documentar el despliegue.

Cronograma Preliminar de Actividades (Diagrama de Gantt)

El proyecto se planifica para una duración de 11 semanas.



Análisis de requisitos

* Módulo de Reservaciones: Permite crear, consultar, modificar y cancelar reservaciones. Incluirá un calendario visual de ocupación.
* Módulo de Recepción (Front Desk): Para gestionar el proceso de check-in y check-out de los huéspedes.
* Módulo de Gestión de Habitaciones: Administra el estado de las habitaciones (disponible, ocupada, en limpieza, en mantenimiento).
* Módulo de Gestión de Huéspedes: Mantiene una base de datos de clientes con su historial de estancias.
* Módulo de Facturación Básica: Genera facturas simples por estancia, incluyendo cargos de alojamiento.
* Autenticación y Roles de Usuario: Sistema de inicio de sesión para el personal con roles definidos (ej. Recepcionista, Administrador).

Funcionalidades que NO se desarrollarán (Out-of-Scope):

* Integración con canales de venta de terceros (OTAs como Booking.com, Expedia).
* Módulo de punto de venta (POS) para restaurantes o tiendas.
* Sistema de gestión de inventario.
* Funcionalidades avanzadas de marketing (CRM).
* Aplicación móvil para el personal de limpieza (housekeeping).
* Módulo de contabilidad avanzada.

Objetivos Específicos (con métricas)

Los objetivos específicos también definen requerimientos medibles:

* Diseñar e implementar una base de datos relacional para almacenar de forma segura la información.
* Construir un módulo de reservaciones que permita gestionar el ciclo de vida de una reserva en menos de 3 minutos por operación.
* Desarrollar una interfaz de usuario web intuitiva que facilite los procesos de check-in y check-out, reduciendo el tiempo promedio por huésped en un 40% en comparación con un sistema manual.

Análisis de Requerimientos Iniciales

El documento presenta una lista explícita de requerimientos funcionales y no funcionales.

Requerimientos Funcionales (RF):

* RF-001: El sistema debe permitir al recepcionista crear una nueva reservación (especificando fechas, tipo de habitación y datos del huésped).
* RF-002: El sistema debe mostrar un calendario o dashboard visual con la ocupación de las habitaciones por fecha.
* RF-003: El sistema debe permitir modificar los detalles de una reservación existente.
* RF-004: El sistema debe permitir cancelar una reservación, liberando la habitación.
* RF-005: El sistema debe permitir registrar la llegada de un huésped (check-in), cambiando el estado de la habitación a "Ocupada".
* RF-006: El sistema debe permitir registrar la salida de un huésped (check-out), generando una factura y cambiando el estado de la habitación a "En Limpieza".

* RF-007: El sistema debe permitir registrar y consultar la información de los huéspedes.
* RF-008: El sistema debe permitir al administrador cambiar el estado de una habitación manualmente (ej. a "Mantenimiento").
* RF-009: El sistema debe requerir autenticación (usuario y contraseña).
* RF-010: El sistema debe permitir al administrador generar un reporte básico de ocupación por rango de fechas.

Requerimientos No Funcionales (RNF):

* RNF-001 (Usabilidad): La interfaz debe ser intuitiva, permitiendo que un nuevo recepcionista aprenda las operaciones básicas en menos de una hora de capacitación.
* RNF-002 (Rendimiento): Las consultas de disponibilidad de habitaciones deben ejecutarse y mostrar resultados en menos de 2 segundos.
* RNF-003 (Seguridad): La información personal de los huéspedes debe estar encriptada en la base de datos. El acceso debe estar restringido por roles.
* RNF-004 (Disponibilidad): El sistema deberá tener una disponibilidad del 99.5% durante el horario operativo del hotel.
* RNF-005 (Compatibilidad): La aplicación web debe ser compatible con las últimas dos versiones de los navegadores Google Chrome, Mozilla Firefox y Microsoft Edge.

Referencias

1. Arhipov, A. (2024, octubre). **Ktor 3.0 is now available with new features and improved performance**. *JetBrains Kotlin Blog*. https://blog.jetbrains.com/kotlin/2024/10/ktor-3-0/
2. Business Research Insights. (2024). **Property management system (PMS) in hotel market 2024–2034** [Reporte de mercado]. https://www.businessresearchinsights.com/market-reports/property-management-system-pms-in-hotel-market-125031
3. Cockburn, A. (2005, 4 septiembre). **Hexagonal architecture**. https://alistair.cockburn.us/hexagonal-architecture
4. Cook, L. (2025, 16 mayo). **PMS UX explained: How good design transforms hotel operations**. *Hospitality Financial & Technology Professionals*. https://www.hftp.org/news/4127256/pms-ux-explained-how-good-design-transforms-hotel-operations
5. Crudu, A. (2024, 12 marzo). **Enhancing guest experience with cutting-edge hotel management software**. *MoldStud*. https://moldstud.com/articles/p-enhancing-guest-experience-through-innovative-hotel-management-software
6. Devographics. (2024). **State of JavaScript 2024: Usage**. https://2024.stateofjs.com/usage/
7. EnterpriseDB. (2025, 19 febrero). **EDB Postgres AI significantly outperforms Oracle, SQL Server, MongoDB, and MySQL in new benchmark study**. *EnterpriseDB Press Release*.

https://www.enterprisedb.com/news/edb-postgresr-ai-significantly-outperforms-oracle-sql-server-mongodb-and-mysql-new-benchmark

8. Hu, M., & Wu, T. (2025, 1 junio). **Constant-factor algorithms for revenue management with consecutive stays** [Preprint]. *arXiv*. https://arxiv.org/abs/2506.00909
9. Jishan, M. A., Singh, V., Ghosh, A. K., Alam, M. S., Mahmud, K. R., & Paul, B. (2024, 21 octubre). **Hotel booking cancellation prediction using applied Bayesian models** (Version v2) [Preprint]. *arXiv*. https://arxiv.org/abs/2410.16406
10. Market Growth Reports. (2024). **Hotel and hospitality management software market size & forecast 2025-2033**. *Market Growth Reports*. https://www.marketgrowthreports.com/market-reports/hotel-and-hospitality-management-software-market-110773
11. Market Growth Reports. (2025, 8 septiembre). **Hospitality property management software (PMS) market size, share, growth, and industry analysis to 2033**. *Market Growth Reports*. https://www.marketgrowthreports.com/market-reports/hospitality-property-management-software-market-104797
12. Mulet-Forteza, C., Ferrer-Rosell, B., Martorell Cunill, O., & Linares-Mustarós, S. (2024, 7 noviembre). **The role of expansion strategies and operational attributes on hotel performance: A compositional approach** [Preprint]. *arXiv*. https://arxiv.org/abs/2411.04640
13. Oracle. (2023, 11 mayo). **Oracle teams with Wyndham to bring OPERA Cloud to 2,000 additional hotels**. *Oracle Newsroom*. https://www.oracle.com/news/announcement/oracle-teams-with-wyndham-to-bring-opera-cloud-2023-05-11/
14. Oracle. (2023, 27 junio). **Hotels optimize operations and guest experiences with OPERA Cloud**. *Oracle Newsroom*. https://www.oracle.com/news/announcement/hotels-optimize-operations-and-guest-experiences-with-opera-cloud-2023-06-27/
15. Oracle. (2023, 7 noviembre). **EOS taps Oracle Cloud to supercharge hotels and resorts**. *Oracle Newsroom*. https://www.oracle.com/news/announcement/eos-taps-oracle-cloud-to-supercharge-hotels-and-resorts-2023-11-07/
16. Oracle. (2025, 14 agosto). **Oracle named a leader in IDC MarketScape worldwide hospitality property management systems 2025 vendor assessment**. *Oracle Newsroom*. https://www.oracle.com/news/announcement/oracle-named-a-leader-in-idc-marketscape-worldwide-hospitality-property-management-systems2025-vendor-assessment-2025-08-14/
17. Pons, S. T., Noone, B., & Johns, M. (2024). **Integrated revenue strategy for the Park Hotel: A data-driven approach to strategy development and evaluation**. *Journal of Hospitality & Tourism Cases*, *13*(1), 29-44. https://doi.org/10.1177/21649987231221476
18. Quan, S., Tan, H., Liu, S., Zheng, Z., Zhu, R., Li, L., Lu, Q., & Wu, F. (2025, 10 junio). **MERIT: A merchant incentive ranking model for hotel search & ranking** [Preprint].

arXiv. https://arxiv.org/abs/2506.08442

19. Starfleet Research. (2023, 25 mayo). **The 2023 Smart Decision Guide to hotel property management systems**. *Hotel News Resource*. https://www.hotelnewsresource.com/article126536.html
20. Vergauwen, S. (2025, junio). **Ktor 3.2.0 is now available**. *JetBrains Kotlin Blog*. https://blog.jetbrains.com/kotlin/2025/06/ktor-3-2-0-is-now-available/